Parallel Computing for Information Retrieval: Recent Developments

Craig Stanfill

Thinking Machines Corporation
Technical Report Series

DR88-1
1/88

Parallel Computing for Information Retrieval: Recent Developments

Craig Stanfill
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142

ABSTRACT

Parallel computers are likely to be an increasingly important technology for searching text databases. In this paper we consider recent developments in a text retrieval system based on Thinking Machines Corporation's parallel computer, the Connection Machine System®. In particular, we look at algorithmic changes which take advantage of the CM-2, the current model of the machine. These new algorithms offer higher performance than was previously available. We also consider some objections raised in the literature to our performance claims for the machine. Careful benchmarking reveals that, for important classes of retrieval applications, the Connection Machine is, indeed, faster than other alternatives by a factor of 50-80.

1. Introduction

Parallel computers, such as the Connection Machine System®, are emerging as an important technology in the searching of large text databases. This is because the growth in the size of databases is outstripping the growth in processing power available with sequential machines. In addition, as databases become larger the retrieval algorithms needed to locate information may become computationally more difficult. We have argued that the use of parallel computers represents the best means of coping with this explosion in the computational requirements for IR.

We first reported our experience with the Connection Machine System (Model CM-1) in [1]. The basis of this work was a parallel implementation of an overlap-encoding search algorithm. Using it, we proposed an interactive system using relevance feedback applied to a smaller database (32 MB). The primary advantage was high performance and high quality search using relevance feedback.

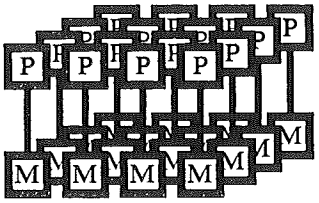
Since then, a second model of the Connection Machine System (the CM-2) has been introduced by Thinking Machines Corporation. In addition, we have implemented

some new variations on our search algorithms. Finally, Stone [2] and Salton [3] have questioned our performance claims. In this paper, we would like to explain the changes incorporated in the new model of the CM, discuss their impact on the design of our retrieval systems, and address the performance issues raised by Stone and by Salton in light of these new developments.

2. The Connection Machine System

2.1. Basic Architecture

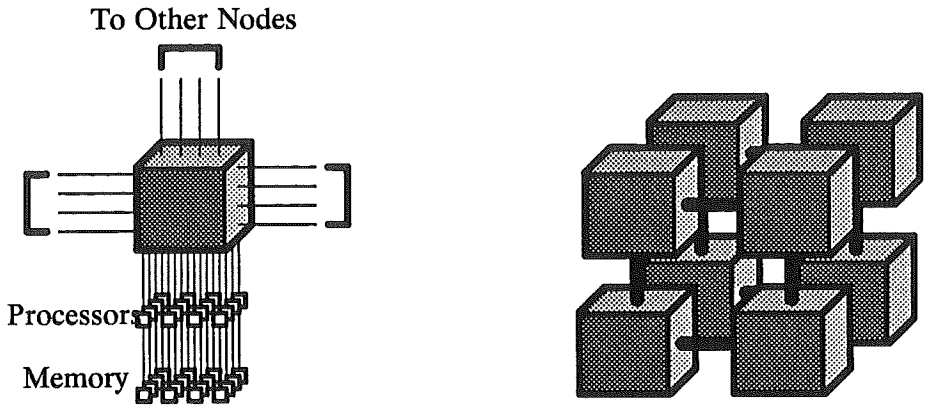
The Connection Machine System is a parallel computer constructed from between 16,384 and 65,536 bit-serial processing elements [4]. Taken together, these processors can perform 128 billion single-bit operations, or 4 billion 32-bit operations, per second. These processors run as a SIMD system, with all processors running the same program simultaneously.



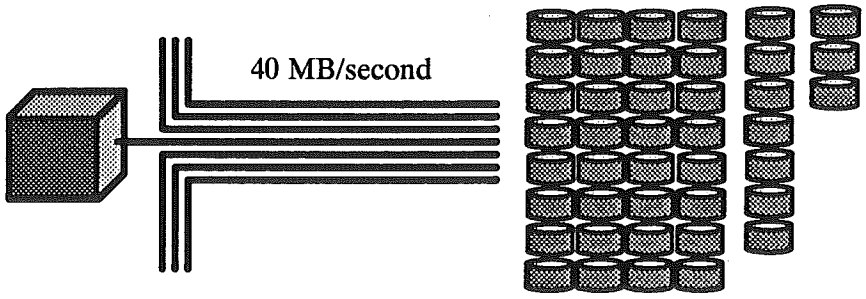
65,536 Bit-Serial Processors

128 Billion	Bit-Operations
4 Billion	Word-Operations
4 Billion	Bits of Memory
512 Million	Bytes of Memory

Each group of 16 processors is integrated with a router node, of which there are 4096 in a 64K-processor system. This provides switching capacity for 65 million messages per second, with even higher performance realized on certain regular patterns. The nodes are wired in a 12-dimensional hypercube.



Finally, there is an I/O system. Each router node is attached to an I/O line. The I/O lines from a large number of chips are multiplexed down to an I/O bus, which may connect to several devices, including the DataVault® disk system. The DataVault is a mass-storage system constructed from 32 Winchester disks plus 7 ECC drives plus 3 spares. At any time, all drives will be reading the same sectors of their respective disks. ECC is employed on a cross-drive basis, so that one gets 32 blocks of data from the primary drives and 7 blocks of ECC from the ECC drives. Should any primary drive fail, the data on the unavailable drive may be reconstructed from the 31 remaining data drives plus the 7 ECC drives. This results in greatly enhanced system availability.



2.2. Recent hardware developments

In part the work reported here has been driven by the introduction of the Connection Machine System model CM-2 [5], a machine markedly improved over the CM-1.

Among the changes incorporated in this new system are:

Increased memory capacity. Whereas the CM-1 had 4096 bits per processing element, the CM-2 has 65,536 bits per element, a 16-fold increase. This increased memory capacity permits the interactive searching of much larger databases than was previously possible.

DataVault Disk System. The disk system mentioned above is now available as a product. Each DataVault has a storage capacity of 5-10 GB, with a sustained transfer rate of 40 MB/second and an average latency of 33 ms. This is critical to the batch search algorithm.

CM I/O Bus. The I/O system of a 64K-processor CM-2 allows for up to 8 simultaneously active I/O busses. Assuming that all are connected to DataVaults, this gives a sustained transfer rate of 320 MB/second. This is critical to the batch search algorithm.

3. Background

3.1. Boolean Search Systems

Most retrieval systems in commercial use are based, directly or indirectly, on systems employing inverted indexes and boolean queries. The retrieval strategies generally used with such systems employ boolean queries hand-formulated by the user of the system. These systems will initially return a count of how many items matched the query; the user will then either reformulate his query based on the hit-count or read some of the retrieved items.

On the one hand, systems of this type tend to be relatively modest in their computational demands: queries are usually small (2-10 terms), and the inverted index technique makes good use of the limited I/O and computational capabilities of serial machines. On the other hand, formulating boolean queries by hand is not the best method of retrieving information and, in fact, users of such systems find them difficult to use.

3.2. SMART

The SMART system [6] represents an advance over STAIRS-like systems in terms of its retrieval strategy. First, it is based on the vector model: documents and queries are vectors which assign weights to various terms. A wide variety of methods have been

proposed for computing and normalizing these weights [7]. Second, retrieval consists of scoring document–vectors as to how well they match query–vectors, then returning the top–ranked documents to the user. This score–and–rank method of retrieval generally yields superior quality of search while, at the same time, being easier to use. Finally, relevance feedback (rather than manual query reformulation) is used to improve retrieval effectiveness from iteration to iteration. In relevance feedback, the user of the system will indicate that certain of the retrieved documents are "relevant;" the system will then extract the important terms from these documents and add them to the user's query. Relevance feedback has been shown to significantly improve search performance.

One difficulty with relevance feedback is that it may produce rather large queries: 100 terms is not uncommon, in our experience. If the search engine is not sufficiently powerful in relationship to the database size, the effort required to evaluate such queries may prohibit (or sharply limit) the use of this technique.

4. CM Retrieval System

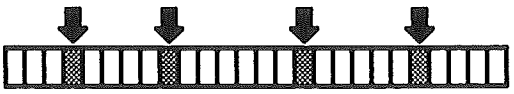
Parallel search techniques [1] represent an advance in the performance of the retrieval systems. This is important in the context of growing database size and of the increased computational demands posed by relevance feedback techniques. Over the last two years, we have been studying a variety of signature–based techniques for information retrieval on the Connection Machine System. This section will present the most important of these methods.

4.1. Representation

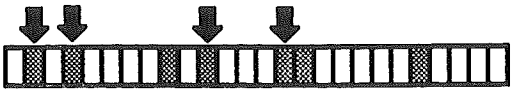
The basic representation consists of storing several overlap–encoded signatures in each of the Connection Machine's processors. We can then probe all signature for the presence of a word in time proportional to the number of signatures per processor.

Briefly, the overlap encoding involves initializing a table of bits to 0:

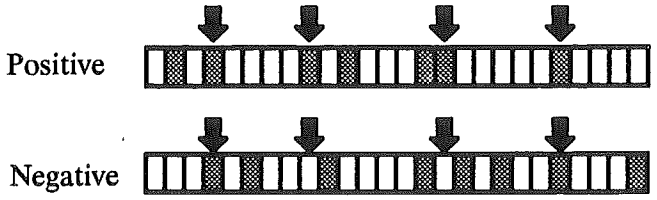
To insert a word, a number of hash–codes are generated for that word, and the corresponding bits set to 1.



To insert additional words, the process is repeated.



To test for the presence of a word, the same hash functions are generated, and the corresponding bits of the table AND'ed together. If all the bits are found to be 1, then the probe returns *positive;* otherwise it returns *negative.* There is always a possibility that a false positive will be returned but, with proper choice of table size and the number of hash codes, this possibility may be rendered so unlikely as to cause no trouble.



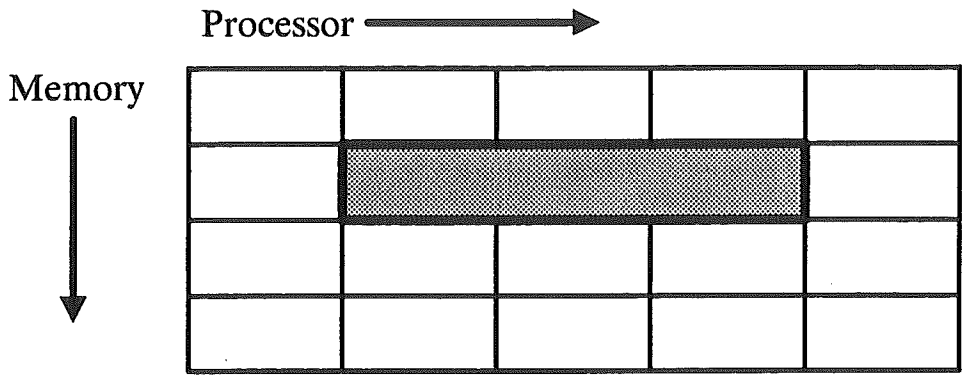
4.2. Signature Size

On the CM–1, memory–size limited us to 3 signatures of 1024 bits per processor. In order to limit the number of false positives returned during probing, we put no more than 30 words into each signature. The CM–2's larger memory allows us room for 54 signatures. This has the advantage of allowing us to store 16x more data, but the disadvantage of slowing searches by a factor of 16. The solution to this problem is to make the signatures larger. By using 4096 bit signatures, the number of signatures per processor is reduced from 54 to 17. The cost of doing so is a slight loss of storage capacity due to fragmentation. Suppose, for example, that we have a document with 130 words. Using 1024–bits signatures, and putting 30 words into each signature, this document completely fills 4 signatures; the last signature then gets the last 10 words, for a total of 5 signatures (5K bits). Using 4096 bit signatures, each receiving 120 words, we would completely fill one signature, then put 10 words into a second, for a total of 2 signatures (8K bits). According to our measurements, this effect increases the storage requirements for a typical database by 20%. The 4–fold increase in performance is, in our opinion, well worth this slight loss of storage efficiency.

4.3. Chaining

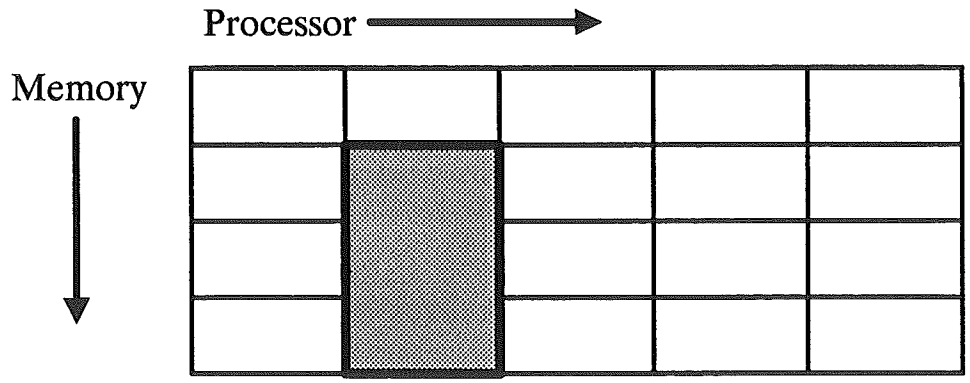
Documents too large to fit into a single signature must be split among several signatures. The signatures are probed individually, then the results of these individual probes combined to form a probe of the entire document.

When probing a document for the presence of a word, it is necessary to combine the results of probes for the several signatures making up the word, OR'ing them together. In the CM–1, the documents were arranged so that the various signatures for a document were stored at the same memory address in adjacent processors:



Combining the results of probes on signatures to produce probes on documents then requires OR'ing together 1 bit each in a group of adjacent processors. It might appear that, if the longest document in a given row of processors consisted of N signatures, then N–1 communication operations would be required to process that row. This is not correct; the Connection Machine implements some low–level communication routines (the so–called "scan operations" [8]) that allow this cumulative–or to be computed in a single operation. Nevertheless, one communication operation is (under this arrangement) required per row of signatures.

By re–arranging the data so that the signatures for a document reside in adjacent memory locations of a single processor, we can substitute local operations (ones within a single processor) for the bulk of these communications operations:



When this is done, we require one local operation per row of signatures plus one communications operation (to handle the cases where a document crosses a processor

boundary). Local operations being faster than communication operations on the Connection Machine, this implementation is favored.

4.4. Applications

From this starting point, we implement both boolean and weighted searches. For weighted searches, we generally use binary document vectors (i.e. words are either *present* or *absent*) but we can, by using multiple signatures per document and inserting the same word in several different signatures, realize an approximation to IDF weighting schemes in which a words may be present with varying strengths, in proportion to the number of times it occurs in the document. However, we usually do not use this method, and its application will not be further discussed at this time.

We have proposed two applications of this algorithm. The first is an interactive system employing relevance feedback designed to search databases in the 512–2048 MB range. The second is a batch system designed to search databases in the 64 GB range.

4.4.1. An Interactive System

The interactive system operates by filling the CM's memory with signatures, then applying weighted queries generated both by the user and by relevance feedback to rank documents in order of probable relevance. To search a 2 GB database on a 64K–processor CM–2, using a 100–term query generated by relevance feedback, should require 72 ms. As a note, our original prototype system ran on a 16K CM–1, and could apply a similar query to a 40 MB database in 60 ms. Use of a machine with more processors, larger signatures, and improved chaining has resulted in a 54–fold increase in performance (13–fold holding the number of processors constant).

As is generally the case for systems employing relevance feedback and weighted queries, this system proved easy to use while delivering high search quality. Issues relating to performance will be discussed below.

4.4.2. The Batch System

The system described above is adequate only for databases which fit into memory. Larger databases require access to secondary storage. For the signature methods we have been using, this involves building a file of signatures, then repeatedly loading memory with signatures and evaluating the query. This system achieves high throughput, but the typical turnaround time of 2–3 minutes for this application relegates it to batch search environments.

4.5. Issues raised in the literature

Stone [2] and Salton [3] have published analyses of these results. Stone primarily analyzed the batch system, and argued that for all but the largest queries the I/O traffic

generated by the batch algorithm renders it inferior to an inverted file system on a sequential computer. Salton analyzed the interactive system, and argued that it was possible to realize equivalent performance using inverted indexes stored entirely in memory. He also objected to some of our specific retrieval methods. In this paper, we will respond to Stone and Salton's questions on performance; questions as to effectiveness of search will be addressed at another time.

5. Performance of the Interactive System

Salton has raised the issue of whether a serial machine having a large amount of memory, using an inverted index algorithm, might perform as well as a Connection Machine System. In particular, Salton claims to have a program running on a SUN-3 which has the same performance as a 16K CM-1, using an inverted index algorithm. He quotes a time of 40 ms to resolve a 200-term query (exclusive of the time to rank the documents after scoring, exclusive of I/O). This is indeed the same as the time required on a 16K CM-1, under certain assumptions.

In comparing the performance of inverted index and signature methods, one critical factor is the number of documents in which each query-term occurs. This is because the amount of time required by the signature algorithm is independent of the number of documents in which a query-term occurs, whereas the indexed algorithm requires time proportional to the number of term occurrences. Thus, the proportion of documents in which an average term occurs (we will call this the *posting rate*) is of critical importance. A high posting rate favors the signature algorithm, a low rate favors the indexed algorithm. We will, therefore, begin with a discussion of posting rates.

The first thing to remember about posting rates is that they depend on both the database being studied and on the source of the queries. There being no better alternative, we have measured the posting rate for the sort of database and query of interest to us. The database here is *The Wall Street Journal.* It tends to have a mixture of article lengths, with the average being around 4096 bytes (256 words excluding stop-words). There are, however, a significant number of very short articles (64 words and fewer), and a small number of very long articles (2000 words and up). Understanding the mix of document lengths in a database is important because, in general, databases with short articles should have lower posting rates than databases with longer articles. The query source is relevance feedback. Relevance feedback operates by extracting words from documents and using them to build queries. This process tends to put fairly common words into the queries, but even after words which are too common to be of any use (in this example, such words as "Inc." and "Co." in addition to the usual stop-words) are pruned out, the queries contain a relatively high proportion of common words. The result is a measured posting rate of 3%. Measurement of a second database, *Time* magazine, yielded a posting rate of 5% (this higher rate is probably due, in part, to the preponderance of long articles in *Time*). Other databases might have dif-

ferent characteristics; in particular we suspect that bibliographic databases (including abstracts), with their technical vocabulary, would tend to have a lower posting rate than we have observed in journalistic sources.

In Salton's analysis of the CM-1 system, he used a posting rate of 1%, which he got from Waltz [9] . Waltz had stated a posting rate of 1% "per document." That was, however, a misstatement: a careful reading of the text shows that his computation was actually assuming a posting rate of 1% *per signature*. As there are multiple signatures per document, the actual per-document posting rate should be higher. This accounts for part of the differences in computations of the relative merits of the CM-1 and sequential indexed algorithm. Thus, for the databases we are studying a 16K processor CM-1 should be 3x faster than a serial machine, and a 64K processor CM-1 should be 12x faster. However, for some databases Salton's figures may well be correct.

Salton's comparison of the sequential indexed algorithm and our parallel signature algorithm is incomplete in that it only includes the time required to score the documents, and omits the time required to rank them. If he had to sort every document in the database in order of decreasing score, we suspect this step would become the dominant cost. However, we feel it suffices to retrieve the best 20 documents, and will proceed under this assumption.

The algorithm we used for ranking was a binary tree insertion algorithm, keeping as a threshold the score of the 20'th best document seen so far; only documents with scores greater than this threshold were inserted into the tree. The inner loop of this algorithm is somewhat more complex than that of some other algorithms, but it has an attractive computational order.

In order to judge the relative merits of the CM-2 and sequential machines on the complete retrieval task, we ran a benchmark. First, we built a series of queries on our CM retrieval system using relevance feedback. Second, we built an inverted term list sufficient to evaluate each query. We loaded these term lists into a 10-MIP SUN-4, and used them to score the documents in a 13 Mbyte database and rank the 20 best documents. The average time to perform this task for a query of approximately 100 terms was 37 milliseconds. The ranking algorithm accounted for 15 milliseconds, and query-term evaluation accounted for the rest. Extrapolating linearly to a 2 GByte database, we get a time of 5.6 seconds. A 64K-processor CM-2, using the improved algorithms noted above, can search a 2 GB database in 72 milliseconds, for an 80-fold speedup.

Salton questions whether any speedup is necessary in the first place. One can argue that 5.6 seconds is a marginally acceptable response time for an interactive system. However, this difference in response time translates directly into a difference in the number of users the system can support: a machine which is 80 times faster can support 80 times the user community. Also, it must be remembered that a turnaround time of 5.6 seconds will be realized only on a completely unloaded system; as the system becomes heavily loaded queuing delays can easily double or triple the average waiting time.

6. Performance of the Batch Algorithm

The interactive system shown above is adequate only for databases which fit completely in memory (2 GB, at present). For larger databases, we do not at present have an interactive system, but we have designed a batch system with a projected turnaround time of 2–3 minutes. This system operates by repeatedly loading segments of the signature file into memory and searching them. Stone [2] has argued that this algorithm is inferior to the indexed algorithm with disk. He traces this difference in speeds to the I/O traffic generated by the batch signature algorithm: the signature method must read the entire file into memory, while the indexed algorithm need only read the indexes for those terms which are referenced in the query (he does not dispute our claim that the in-memory part of the CM algorithm is faster).

The I/O done by the parallel algorithm is constant, while that done by the indexed algorithm is proportional to the number of terms in the query. It is clear that, for sufficiently short queries, the indexed algorithm does less I/O, and that for sufficiently long queries, it does more. The question is: where is the cross-over point? At this point our analysis differs from Stone's, again on the critical point of the posting rate. Stone assumes that there are 65K different words in the database, and that queries contain these words with equal probability. In our experience this is not at all the case: queries — particularly those constructed via relevance feedback — contain a disproportionate number of common terms. The result is that Stone's model predicts a posting-rate of 0.2%, as opposed to our measured rate of 3%. Taking this into account, but otherwise using Stone's computations, the break-even point in terms of total I/O moves from queries of 65,000 terms to queries of approximately 4000 terms.

This may not at first seem significant, because even with relevance feedback one does not encounter queries of 4000 terms. However, remember that this is a batch, not an interactive algorithm, and that several small queries may be combined in a single run. For example, if we have 40 users, each of whom submits a 100-term query, we get a batch with 4000 terms. Such a batch-job is processed by loading the CM's memory with signatures, running all 40 queries on the contents of memory, ranking the documents so found, and saving the 40 query-results so found. At the end of the run, these results may be merged in a few tens of milliseconds.

Let us assume a database with the following characteristics (following Stone, but using our own figures for the posting rate):

Documents	32 Million
Document Size	128 words
Database Size	64 Gbytes
Terms/Query	100
Posting Rate	3%

As a note, these database statistics are slightly different from those observed in our *Wall Street Journal* database; the average size of the documents here is half what was

observed in the WSJ database. Again following Stone, there are 32 M * 128 = 4 G word-document pairs in the database. As each 4096-bit signature holds 128 words, this requires 32 M signatures, which is 16 Gbytes of storage (ignoring a small penalty for fragmentation). We process this job by loading the CM's memory with data, running the 40 queries, and repeating 32 times (thus processing 2 Gbytes worth of the initial database in each memory-load). Each of these 32 search-phases requires us to load 500 Mbytes data (16,000 / 32), and to run 40 queries of 100 terms each. Using the CM's full I/O capacity of 320 MB/second, the I/O time for a search phase is 1.56 seconds. Using the figure of .072 seconds to run a 100-term query (reported above), we get a per-phase compute time of 2.88 seconds. Thus, each phase requires a total of 4.44 seconds, and the full run of 32 phases requires 142 seconds.

For the inverted index method, we also have 100 terms/query. We will first estimate the compute time. As noted above, a 10 MIP serial machine requires 5.6 seconds to resolve a 100-term query against a 2 Gbyte database. Extrapolating to the 64 Gbyte database we are working with here, we get a compute time of 179 seconds. We now turn to I/O. As noted above, our figures indicate that each term will have postings in 3% of all documents. This yields 960K postings per query-term, and a total of 96M postings per query. Each posting generates 4 bytes of I/O traffic, for a total of 384 Mbytes I/O. Assuming I/O overlaps with computation, an I/O rate of only 2 Mbytes/second is required, and we may assume the sequential machine spends 100% of its time computing.

Thus, we see that the CM has slightly better response time and much better throughput: in 142 seconds, the CM returns the answer to 40 queries, while a sequential machine requires 179 seconds to answer just one.

Note that in this analysis, I/O time is not as much of an issue as Stone would suggest. The sequential processor, even running the "efficient" indexed algorithm, is so compute-bound that high I/O performance is not required. On the other hand, by batching queries on the CM we can achieve a balance between computation and I/O, at which point compute time becomes the primary issue: the CM spends 50% of its time computing, and while the sequential machine spends 100% of its time computing this can not make up for the 80-fold difference in speeds.

7. Summary

Our contention throughout this debate has been that the growth in on-line databases makes the utilization of parallel computers essential. We have developed two signature-based algorithms: an interactive system which stores signatures in memory, and a batch system which repeatedly loads the CM's memory with signatures and searches it. Our conclusions on these matters are as follows:

1. The interactive system, running on a 64K-processor CM-2 and using our current algorithms, can hold a database of up to 2 Gbytes. On the 100-term rele-

vance feedback queries we have been using, it yields 80 times the performance of a 10 MIP serial machine running the sequential indexed algorithm (including the time to rank the documents).

2. The batch system running on a 64K-processor CM-2 with a full I/O system can execute a batch of 40 queries of 100 terms each on a 64 Gbyte database in 142 seconds. This compares with 179 seconds for a 10 MIP serial machine executing a single query. This represents slightly better turnaround time and 50 times higher throughput.

This is not to say that these algorithms are correct for all applications. For example, for 5-term queries applied to a 64 Gbyte database in an interactive environment, the indexed algorithm with disk will have a response time near 9 seconds, and while the parallel batch algorithm might have greater throughput, its minimum response time of 142 seconds would render it unsuitable. Unfortunately, there are important applications for which there does not, at present, appear to be a good solution. For example, we are not aware of any means of applying 100-term relevance feedback queries to 64 Gbyte databases with interactive response times.

We must caution the reader that all comparisons between indexed algorithms and signature algorithms depend critically on the posting rate which depends, in turn, on the database and the queries being processed.. If, for example, the posting rate for a set of queries and some database is around 0.1% (as Stone assumes), the parallel signature method's performance advantage evaporates, and serial indexed algorithms are indicated.

Finally, this is not to say that the signature method is the best possible parallel algorithm. For example, Stone has suggested that a parallel indexed algorithm might be superior to the parallel signature algorithm. We are very interested in such algorithms, but until we have benchmarked them we have no way of saying how fast they really are.

In any event, parallel document retrieval algorithms are certain to be of growing importance in coming years. For important applications, they are already the fastest method by a factor of 50-80, and as the technology evolves the range of applications for which suitable parallel techniques exist can only increase.

REFERENCES

[1] Stanfill, C., and Kahle, B., "Parallel Free-Text Search on the Connection Machine System," *Communications of the ACM*, Volume 29 Number 12, December 1986, pp. 1229-1239.

[2] Stone, H. S., "Parallel Querying of Large Databases: A Case Study," *IEEE Computer*, October 1987, pp. 11-21.

[3] Salton, G. J., "Parallel Text Search Methods," Technical Report 87-828, Department of Computer Science, Cornell University, April 1987.

[4] Hillis, D., *The Connection Machine,* MIT Press, Cambridge MA, 1985.

[5] "CM-2 Technical Summary," Thinking Machines Corporation Technical Report, 1987.

[6] Salton, G.J., *The SMART Retrieval System — Experiment in Automatic Document Processing,* Prentice-Hall, Englewood Cliffs, NJ, 1971.

[7] Salton, G.J., "Term Weighting Approaches in Automatic Text Retrieval," Technical Report 87-881, Department of Computer Science, Cornell University, November 1987.

[8] Blelloch, G., "Scans as Primitive Parallel Operations," Thinking Machines Corporation Technical Report DP86-1, January 1986.

[9] Waltz, D., "Applications of the Connection Machine," *IEEE Computer,* January 1987, pp. 85-97.